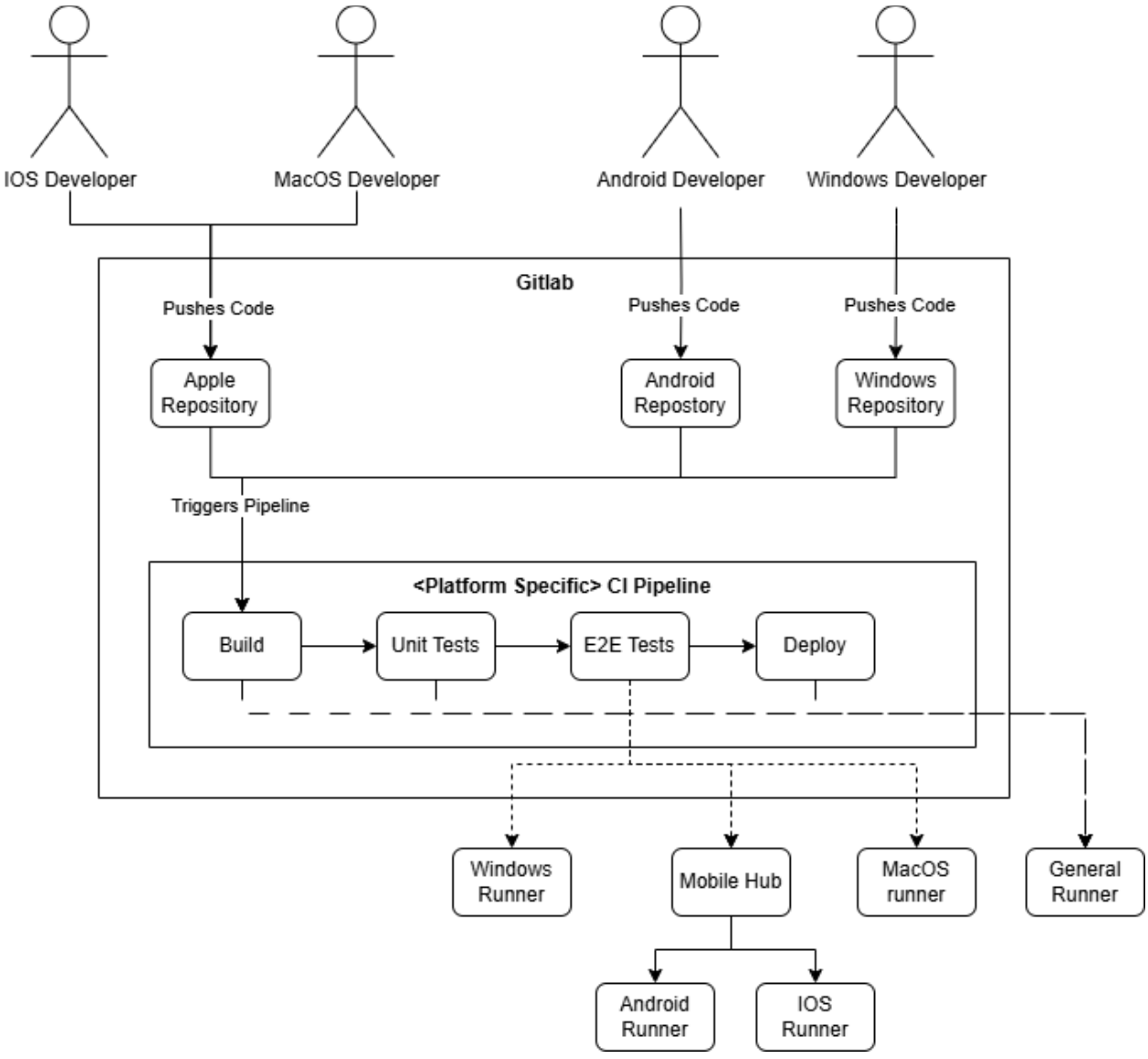
Homework

Part 1:

Disclaimer this is high level description. During the interview with more context we can discuss more in-depth specifics and maybe different approach, when new context is provided.

I'd use CI/CD pipelines. For code storage and runner implementation I'd use Gitlab. Since Gitlab already has effective version control and runner implementation. The flow would look like this:



The flow starts from developer pushing the code. Every platform has it‘s own repository so that the code would not conflict with others. It‘s worth noting that MacOS and IOS developers are sharing same repository, I‘m assuming that SwiftUI is used and this framework allows seamless integration of both platforms in one repository.

Second step is pipeline trigger. It really depends on the team processes and duration of tests. But I‘d go with pipeline trigger for every merge request. **Every platform‘s repostiory** would have to contain **Build, Unit Tests, E2E Tests, Deploy** jobs in it‘s pipeline. Build, Unit tests and Deploy jobs would have to share General runner. The reason is that usually those jobs doesn‘t require platform specific tools. And this would allow us to easily scale, while maintaining reasonable amount of runners. Docker would be used to spin up and duplicate those general runners. This runner would run some Unix OS, e.g Ubuntu, since it‘s easily scalable.

Then there is E2E tests, different platforms would have to contain platform specific runers, since it‘s impossible to run UI tests on one platform.

- Windows platform would use virtualization tools e.g **HyperV** or **Packer.** Since virtualization for Windows is quite mature and doesn‘t have that many limitations.
- Mobile runners have to be connected to one central mobile hub, which could be running Ubuntu. This Mobile hub is needed for gitlab runner communication between mobile devices. It‘s impossible to set up CI/CD runner inside mobile devices. Virtualization options could be explored for Android, but for iOS we can‘t, since Similators does not support VPN connections.
- MacOS runner would be a real device, since virtualization is not that scalable on this platform.

Pros:

- Flow is fully automatized after pushing the code.
- The number of runners is easily scalable
- Redundancy can be implemented, to prevent major outages
- Version control processes is smooth in Gitlab

Cons:

- This setup is complex, so it requires good DevOps knowledge.
- Short term costs are high.
- Maintenance is considerable, since different OS runners are implemented.

Comparison with other approaches:

More Centralized CI/CD pipelines

Goal of this approach is to get rid of duplication and try to move common logic to shared scripts. This would get rid of paltform specific CI/CD pipelines and one tempalte could handle CI processes. The reason why I did not chose this approach, even if it sounds good on paper, this approach would reduce flexibility of different platform teams. It'd be hard to implement custom solutions for different platforms. It'd also require careful synchronization between OS teams. Which would slow down releases and reduce velocity.

Infrastructure as a code

This approach is very effective in maintaining testing or any other infrastructure. The main idea is like name suggest you define infrastructure in code, you configure it how you want. Which allows you to have streamlined infrastructure deploys. But the problem is that this approach is meant for infrastructure maintenance, not full code release process.

Part 2:

https://github.com/saka-lukas/nord-homework

https://lukasmakas.grafana.net/public-dashboards/c8465c545b124c88b3cc1c6a0cae7f65